Application of a Multi-Processor SoC Platform to High-Speed Packet Forwarding

Pierre G. Paulin, Chuck Pilkington,
Essaid Bensoudane, Michel Langevin, Damien Lyonnard
Central R&D, STMicroelectronics, Ottawa, Canada
pierre.paulin@st.com

Abstract

In this paper, we explore the requirements of emerging complex SoC's and describe StepNP, an experimental flexible, multi-processor SoC platform targeted towards communications and networking applications.

We present the results of mapping an internet protocol (IPv4) packet forwarding application, running at 2.5Gb/s and 10Gb/s. We demonstrate how the use of high-speed hardware-assisted messaging and dynamic task allocation in the StepNP platform allows us to achieve very high processor utilization rates (up to 97%) in spite of the presence of high network-on-chip and memory access latencies. The inter-processor communication overhead is kept very low, representing only 9% of instructions.

1. Introduction

The continued increase in the non-recurring expenses (NRE) for the manufacturing and design of nanoscale systems-on-chip (SoC), in the face of continued time-to-market pressures, is leading to the need for significant changes to their design and manufacturing. As discussed in [1], [2], these factors are the drivers behind the emergence of domain-specific S/W programmable SoC platforms consisting of large, heterogeneous sets of embedded processors, reconfigurable H/W and networks-on-chip (NoC) [3]. Industrial case studies justifying the use of various platform components were described in [4].

2. Survey of Multi-Processor SoC Platforms

A number of multi-processor platforms designed for SoC-scale applications have been described. Daytona [5] was an early attempt to reach high DSP performances through MIMD processing elements (PE). Each PE consists in a 32b GP-RISC and a vector unit with four 16b-MAC's. The performances reach a peak value of 1.6 billion 16b-MAC/s, assuming no cache misses. Such results are extremely dependent on the instruction locality and require homogeneous data stream rates. This would not be expected for applications that are more control dominated.

The PROPHID [6] based platform, namely Eclipse [7], has already been tuned into several dedicated instances. Among them is the well-known Viper [8] that provides set-top boxes applications with relevant multimedia features. Unfortunately, the use of numerous application-specific hardware accelerators inevitably leads to the high NRE costs of ASIC-style design.

The MESCAL system [9] allows a platform designer to build a platform instance in a targeted, domain-specific way. This is achieved through a range of activities – spanning PE architecture and micro-architecture design, to network topology definition – achieved with the assistance of the Mescal development system [10]. An OSI-like message passing model [11] is used. While this approach may be used to achieve the best cost/performance trade-off, it still implies high design and maskset NRE's.

S^3E^2S [12] is a design environment for heterogeneous multi-processor architectures based on libraries of components. A sequential model of the application is first translated in a CDFG-like objects graph. Then each object is targeted to the most relevant processor selected from the libraries. The design-space exploration addresses the CPU's choice (ranging from GP CPU, DSP to highly specific ones) while taking into account the local memory accesses. Message-passing is the only supported communication mechanism. Also, the NoC topology and implementation is not addressed (nor modeled).

3. StepNP, a Domain-specific multi-processor SoC platform

Figure 1 depicts the *StepNP™* flexible multi-processor architecture platform. The StepNP platform includes models of configurable processors, a network-on-chip, configurable H/W processing elements, as well as networking-oriented I/O's. It embodies what we believe are the key features of emerging MP-SoC platforms in order to address the requirements for flexibility, rapid platform development and platform end-user productivity.

3.1 StepNP Processors

It is our conviction that the large-scale use of software programmable embedded processors will emerge as the key

Proceedings of the Design, Automation and Test in Europe Conference and Exhibition Designers' Forum (DATE'04)
1530-1591/04 $20.00 © 2004 **IEEE**

means to improve flexibility and productivity. These processors will come in a wide diversity, from general-purpose RISC to specialized application-specific instruction-set processors (ASIP), with different trade-offs in time-to-market versus product differentiation (power, performance, cost). Domain- or application-specific processors will play an important role in bridging the gap between the required ease-of-use and high flexibility of general-purpose processors on one end, and the higher speed and/or lower power of hardware on the other. Configurable processors are one possible means to achieve processor specialization from a RISC-based platform.

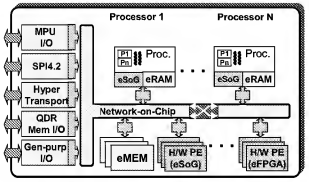

Figure 1. *StepNP* MP-SoC Platform

A common requirement for all classes of processors is the efficient handling of the latencies of the interconnect, memory and co-processors. A variety of approaches can be used. These include caching, multi-threading, memory pre-fetching, and split-transaction interconnect. Multi-threading lets the processor execute other streams while another thread is blocked on a high latency operation. A *hardware multithreaded* processor has separate register banks for different threads, allowing low-overhead switching between threads, often with no disruption to the processor pipeline.

The StepNP simulation framework allows easy integration of a range of general-purpose to application-specific processor models. We have integrated public domain instruction-set models of the most popular RISC processors. To explore various hardware multi-threading and pipeline depths, we encapsulate the functional instruction-set models into a SystemC model wrapper. This produces a cycle-based model implementing a configurable hardware multithreading capability and a simple *n*-stage pipeline, as described in [13].

In order to explore network-specific instruction-set optimizations, the Tensilica Xtensa™ configurable processor model [14] has been integrated by our academic research partners [15].

The first planned StepNP platform implementation will use a combination of configurable and reconfigurable processors, using configurable embedded sea-of-gates (eSoG) and embedded field-programmable gate arrays (eFPGA), as explained further.

3.2 StepNP Interconnect

The StepNP platform makes a very important assumption on the interconnect topology: namely, it uses a single interconnect channel that connects all I/O and processing elements. An orthogonal, scaleable, interconnect approach with predictable bandwidth and latency is essential:

1. It provides a regular, plug-and-play methodology for interconnecting various hardwired, reconfigurable or S/W programmable IP's.
2. It supports the high-level communication between processes on multiple processors, and simplifies the automatic mapping onto the interconnect technology.

However, it moves the complexity of the effective use of communication resources to the resource allocation tools, which must be tuned to the interconnect topology.

We advocate the recent so called 'network-on-chip' (NoC) approaches currently under development [3]. Our first interconnect channel used in StepNP was based on transaction-level models (TLM) of a communication channel using the OCP-IP protocol [16], as discussed in [13]. The first planned implementation of the StepNP platform will be based on an ST interconnect technology, the STBus [17]. The STBus protocol supports similar advanced features to OCP-IP, for example, out-of-order split-transactions. Despite the name, STBus is not a bus per se, but is in fact an interconnect generation framework, which supports the automatic generation of a range of interconnect topologies made up of buses, bridges and crossbars. The STBus toolset generates an RTL-synthesizable implementation. We have integrated the STBus SystemC model into StepNP. Other NoC approaches are also being investigated:

- In cooperation with the UPMC/LIP6 laboratory in Paris, we have developed a 32 port version of the SPIN network-on-chip [18], implemented using ST's 0.13 micron process.
- A ring-based NoC topology is also under development. This provides high scalability and can be designed as non-blocking, but at the expense of higher latencies.
- Finally, for the emerging 65nm process technology node and beyond, we are exploring globally asynchronous, locally synchronous approaches. One interesting example of this approach is the Star network, which serializes packets and uses plesiochronous clocking regions [19].

A common issue with all NoC topologies is communication latency. In 50nm process technologies, it is predicted that the intra-chip propagation delay will be between six and ten clock cycles [3]. Moreover, the increasing gap between processor clock cycle times and memory access times further increases the need for latency

hiding. Effective latency hiding is therefore key in achieving efficient parallel processing.

This is the key reason for the adoption of hardware multi-threading processors in the StepNP platform. This implies that the programming tools must be able to automatically exploit this capability. This was achieved using hardware assisted dynamic task allocation, as described below.

3.3 Embedded FPGA's and Sea-of-Gates

It is our belief that the large majority of end-user SoC product *functionality* will run on the heterogeneous embedded processors. However, power and performance constraints will dictate partitions where the majority of *performance* will come from a combination of optimized H/W, embedded sea-of-gates (eSoG) or embedded FPGA (eFPGA), implementing critical inner loops and parallel operations, but of lower functional complexity.

Embedded FPGA's are used in the StepNP platform to complement the processors, but with limited scope. The ~50X cost and ~5X power penalty of eFPGA's restricts more widespread use. Nevertheless, for high-throughput and simple functions, or highly parallel and regular computations, eFPGA's can play an important role. An eFPGA test chip was developed in ST's 0.18 micron CMOS process, and results were presented in [20]

Embedded SoG technology, e.g. such as that proposed by eASIC [21], which is configured with one or two masks, is another interesting cost and flexibility compromise which we are also incorporating in StepNP. A test chip including this technology was developed in ST's 0.13 micron CMOS process. The 3x to 3.5x cost penalty over standard cells is compensated by the lower maskset NRE, which can be 10x to 30x lower cost than a complete maskset.

3.4 Configurable Processor Implementation

The StepNP platform uses eFPGA's and eSoG in two roles. Their first use is for reconfigurable and configurable processors. ST has developped and manufactured a 1 GOPS reconfigurable signal processing chip [20]. This combines a commercial configurable RISC core with an eFPGA which implements the application-specific instructions. In the StepNP physical implementation, we extend this approach to use embedded sea-of-gates to achieve a low-cost, one-time configurable version of these application-specific instructions.

3.5 Hardware Processing Elements

The hardware processing elements (H/W PE) of StepNP are implementable using a user-defined combination of eFPGA's and eSoG's. To facilitate interoperability, all processing elements communicate to the NoC via a standard protocol. The conversion between the H/W PE's internal data representations and the packet-oriented format of the NoC (as depicted by the 'packetization' blocks of Figure 1), is performed by H/W wrappers automatically generated by the SIDL compiler described below.

The key characteristic of the StepNP platform is that, although it is composed of heterogeneous hardware and software processing elements, memories and I/O blocks, the use of a single standardized protocol to communicate with a single global NoC allowed us to build a *homogeneous* programming environment supporting automatic application-to-platform mapping.

4. MultiFlex Programming Models and Support

The 'MultiFlex' application development environment was developed for multi-processor SoC systems, with networking and communications applications as the first key drivers. Our previous work was concerned mostly with the development of multi-processor modeling, debug and analysis tools [13]. Our current tool developments address parallel programming models and the mapping of a system-level application onto multi-processor and hardware platforms. A brief summary of the supported programming models is provided here.

4.1 MultiFlex Programming Models

Two parallel programming models are used in the MultiFlex system. These models are inspired by leading-edge approaches for large system development, but adapted and constrained for the SoC domain. These two models are:

- Distributed System Object Component (*DSOC*) model. This model supports heterogeneous distributed computing, reminiscent of CORBA and Microsoft DCOM distributed component object models. It is a message-passing model and it supports a very simple CORBA-like interface definition language, dubbed SIDL. Although the SIDL syntax is SystemC-like and therefore very different from CORBA's, it shares conceptual similarities. The SIDL description defines the interface to an object, in a language neutral way. A compiler is used to process this interface, and generate the client or server wrappers in the language of choice.
- Symmetric multi-processing (SMP), supporting concurrent threads accessing shared memory. The SMP programming concepts used here are similar to those embodied in Java and C# and the implementation performs priority scheduling, and includes support for threads, monitors, conditions and semaphores.

Both approaches have strengths and weaknesses, depending on the application domain. Here, objects can be declared as DSOC or SMP and they can be combined in an interoperable fashion. These programming models and their implementation are described in detail in [22]. Here, we focus on the key hardware components of the platform used to support the DSOC distributed message-passing model. The IPv4 packet processing application presented below makes near exclusive use of the DSOC model.

4.2 DSOC Programming Model Support

As we are targeting this platform at high performance applications, such as network traffic management at 10Gbit/sec line rates, a key design choice is the implementation of some of the key multi-processing functions in hardware. Figure 2 illustrates an instance of the StepNP platform, which includes three DSOC functions that are implemented in hardware:

- The hardware *Message Passing* accelerator engine is used to optimize inter-process communication. It translates outgoing messages into a portable representation, formats them for transmission on the network-on-chip, and provides the reverse function on the receiving end.
- The hardware *Object Request Broker* (ORB) engine is used to coordinate object communication. As the name suggests, the ORB is responsible for brokering transactions between clients and servers. Currently, a simple first-come, first-served load balancing mechanism is implemented. The ORB engine allows multiple servers to service a particular service ID. This allows the immediate direction of a client request to an idle server, if one is available.
- The *Thread Manager* is a hardware device that coordinates the creation and synchronization of execution threads. All logical application threads are directly mapped onto hardware threads. No multiplexing of software threads onto hardware threads is done. Requests for new threads are sent to the thread manager, which selects a free hardware thread to serve the request. The cycle-by-cycle scheduling of active hardware threads on a processor is done by the processor hardware, which currently uses a round robin scheduler.

The end result of this hardware-software partition is that we are able to sustain end-to-end DSOC object calls from one processor to another, at a rate of about 10 million per second, using 500 MHz RISC processors.

5. An IPv4 forwarding application

To illustrate the concepts discussed in this paper, we have mapped a MultiFlex model of a complete IPv4 fast-path application of Figure 2 a) onto the multi-processor execution platform depicted in Figure 2 b).

5.1 Networking application framework

Our application software platform makes use of MIT's open source Click modular router framework for the rapid development of embedded routing-application software [23]. Figure 2 a) depicts sample Click modules performing packet classification, discarding, stripping, and queuing.

We have extended the Click IPv4 packet forwarding model to be compatible with the DSOC programming model by encapsulating Click functions in SIDL-based interfaces. The granularity of the partitioning is user-defined and can be defined at the intra-packet and/or inter-packet levels. It is the latter that is most naturally exploited here, due to low inter-packet dependencies.

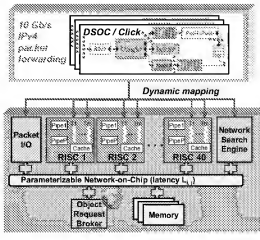

Figure 2. Platform for IPv4 application

5.2 StepNP target architecture

The target architecture platform used here is depicted in Figure 2 b). In order to support wire-speed network processing, a mixed H/W and S/W architecture is used. For example, at a 10Gbps line rate, the packet processing time is approximately 40 nsec. To achieve this, the packet I/O and network address searches are implemented in hardware. The packet I/O component is implemented using 32 hardware threads. It receives input packets, and controls the DMA to memory. The network search engine model is based on ST's NPSE search engine [24]. The platform was configured with the following variable parameters:

- RISC processor ISA: ARM v4
- Number of processor pipeline stages: 4
- Processor clock frequency: 500 MHz
- Data/Program cache size: 4 KB
- Number of processors: 1 to 48
- Number of H/W threads (per proc.): 4 to 32
- One-way NoC latency: 0 to 160 ns
- NoC latency jitter +/- 25%

In this experiment, the NoC latency is a configurable parameter. It represents the total lumped *one way* delay of a data item sent between one master (or slave) to another. We have not attempted to model the detailed behavior of the NoC interconnect, such as contention, blocking, etc. Since the latency value used in the experiment was varied from 0 to 160 ns (or 80 clock cycles at 500 MHz), we believe this is a realistic upper bound on the effect of NoC contention or blocking. Moreover, we include a random jitter on the NoC latency of +/- 25%. This emulates the effect of out-of-order packet processing.

We assume a simple fine-grained multi-threaded processor architecture in which there is a different thread in

each of the pipeline stage; thus, for the 4 pipeline stages we considered, a minimum of 4 threads per processors is necessary to fully use this resource.

5.3 Multi-processor Compilation and Distribution

Figure 2 illustrates the basic compilation and mapping process used in the MultiFlex MP compilation and allocation approach. The three superimposed boxes of Figure 2 a) represents the processing of three different packets. The top-level IPv4 application can be partitioned at two different levels:

- At the *inter-packet* processing level. For IPv4, the inter-packet dependencies are very low. This allows for a high level of natural parallelism. This parallelism is depicted in Figure 2 a) via the overlapping boxes. Each packet processing is assignable to a different thread.
- At the *intra-packet* processing level. This involves cutting the processing graph at basic block boundaries. This is depicted in with the dotted lines illustrating cut points within a single packet processing.

As explained above, the packet I/O and network address search functions are manually assigned to H/W. The remaining IPv4 packet processing is automatically distributed over the RISC processor threads. The hardware object request broker load-balances the input from the I/O object to the IPv4 packet forwarding clients executing on the RISC processors. When a RISC processor thread completes the main IPv4 processing, another call is issued to the I/O object for output transmission.

5.4 IPv4 Results

Some simulation results using a minimum packet length of 54 bytes are depicted in Figure 3. This depicts the packet processing rate obtained, normalized for one processor, while varying the number of hardware threads supported per processor (from 4 to 32) and the *one way* NoC latency (from 0 to 160 ns). Three important observations:

- The highest processing rate achievable for a single processor is approximately 260 Mb/sec. This provides a lower bound of 40 processors to target a system performance aggregate throughput of 10Gb/sec.
- Assuming a perfect, zero latency interconnect, we observe that the 4 threads configuration nearly achieves the highest processing rate. However, realistic latencies cause significant degradation of processor utilization, dropping below 15% for a 160 ns latency.
- As the number of threads is increased from 4 to 32, the NoC latency is increasingly well masked. This results in high processor utilization (as high as 97%), even with NoC latencies as high as 160 nsec.

Profiling of the compiled IPv4 application code running on the ARM shows that 860 instructions are required to process one packet. Approximately one out of eight (108 out of 860) of these instructions lead to a NoC access. This high access rate, which is inherent to the IPv4 application code, highlights the importance of masking the effect of communication latency.

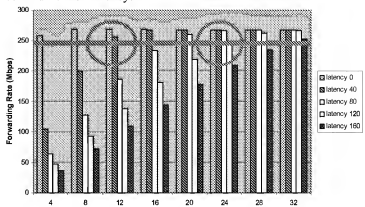

Figure 3. IPv4 Simulation

This example illustrates the importance of the effective utilization of multiple threads in the presence of high latency interconnects. For example, in the presence of a latency of 160 ns, the throughput per processor varies between 40 Mb/s for 4 threads, to 250 Mb/s for 32 threads.

The inter-processor communication represents only 9% of the total packet processing instructions. This very low overhead – especially in this high-speed packet processing context – is achieved using the hardware-based message passing and task scheduling mechanisms described above.

Two representative experimental results are summarized in Table 1. This provides architecture parameters to achieve 2.5Gbps (OC48) and 10Gbps (OC192).

Table 1. IPv4 Simulation Results

Line-rate	#Arm	#thread	NoC latency	ARM utilization	Packet latency
2.5Gbps	16	8	40ns	67%	16us
10Gbps	48	16	80ns	86%	30us

For OC48, a configuration with 16 ARMs with 8 threads each can support the 2.5Gbps line-rate with a NoC latency of 40ns. For OC192, a configuration of 48 ARMs with 16 threads each can support a 10Gbps line-rate. Here, we assumed a higher NoC latency (80ns instead of 40ns) due to the higher number of processing elements. The total packet latency for this configuration is 30us.

In comparison with the 2.5Gbps result, a line rate of 4x is achieved with only 3x processors, in spite of a higher NoC latency (80ns instead of 40 ns). This is a result of the higher processor utilization (86%), which is achieved by using 16 threads instead of 8.

Note that even higher processor utilizations are achievable. With 40 processors and 24 threads, a 97%

processor utilization was obtained. However, this does not offer enough headroom for additional functionality.

For both configurations, 50% of the reported latency to process a packet is a consequence of the NoC latency resulting from the required 108 NoC accesses.

Note that the StepNP platform instance used for this application makes use of standard RISC processors only (ARM v4 ISA). The use of application-specific instructions is not warranted in regular IPv4 packet forwarding since there are very few repeated sequences of operations to optimize. However, our academic partners demonstrated that the use of a Tensilica configurable processor optimized for a secure IPv4 packet forwarding application (using encryption) led to speedups up to 4.55X over an unoptimized core [15]. This example better demonstrates the value of configurable processors in the general StepNP platform.

6. Summary

The *StepNP*^{TM} flexible multi-processor SoC platform consists of multiple configurable hardware multi-threaded processors, configurable hardware processing elements and networking-oriented I/O, connected via a network-on-chip.

Using the MultiFlex *DSOC* distributed message-passing parallel programming model, combined with hardware support for message passing and dynamic task allocation, we achieved a utilization rate of the embedded RISC processors as high as 97%, even in presence of network-on-chip interconnect latencies of up to 160ns, while processing worst-case IPv4 traffic at a 10 Gbps line rate. Only 9% of processor instructions are required for inter-processor communication.

7. REFERENCES

[1] J. Henkel, "Closing the SoC Design Gap", IEEE Computer Magazine, Oct. 2003, pp. 119-121.

[2] P. Magarshack, P.G. Paulin, "System-on-Chip Beyond the Nanometer Wall", Proc. of 40th Design Automation Conference (DAC), Anaheim, June 2003.

[3] A. Jantsch, H. Tenhunen (Eds.), "Networks on Chip", Kluwer Academic Publishers, 2003.

[4] P. G. Paulin et al, "Chips of the Future: Soft, Crunchy or Hard", Proc. of Design Automation and Test in Europe (DATE), Paris, 2004.

[5] B. Ackland et al., "A Single Chip 1.6 Billion 16-b MAC/s Multiprocessor DSP", Proc. of Custom Integrated Circuits Conference, 1999.

[6] J. A. J. Leijten et al., "PROPHID: A Heterogeneous Multiprocessor Architecture for Multimedia", Proc. of Int'l Conference on Computer Design, 1997.

[7] M. Rutten et al., "A Heterogeneous Multiprocessor Architecture for Flexible Media Processing", IEEE Design & Test of Computers, 19(4):39-50, July 2002.

[8] S. Dutta et al., "Viper: A Multiprocessor SoC for Advanced Set-Top Box and Digital TV Systems", IEEE Design & Test of Computers, 18(5):21-31, September 2001.

[9] K. Keutzer, S. Malik, R. Newton, J. Rabaey and A. Sangiovanni-Vincentelli, "System Level Design: Orthogonalization of Concerns and Platform-Based Design", IEEE Trans. on Computer-Aided Design, 19(12), December 2000.

[10] M. Gries, S. Weber and C. Brooks, "The Mescal Architecture Development System (tipi) Tutorial", Technical Report, UCB/ERLM03/40, Electronics Research Lab, University of California at Berkeley, October 2003.

[11] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey and A. Sangiovanni-Vincentelli, "Addressing the System-on-a-Chip Interconnect Woes through Communication Based Design", Proc. of Design Automation Conf., pages 667-672, June 2001.

[12] L. Carro, M. Kreutz, F. R. Wagner and M. Oyamada, "System Synthesis for Multiprocessor Embedded Applications", Proc. of Design Automation and Test in Europe, pages 697-702, March 2000.

[13] P. G. Paulin, C. Pilkington, E. Bensoudane, "StepNP: A System-Level Exploration Platform for Network Processors", IEEE Design & Test of Computers, vol. 19, no.6, Nov. 2002, pp. 17-26.

[14] See Tensilica web site: http://www.tensilica.com

[15] D. Quinn et al, "A System-level Exploration Platform and Methodology for Network Applications Based on Configurable Processors", Proc. of Design Automation and Test in Europe (DATE), Paris, Feb. 2004.

[16] See OCP-IP web site: http://www.ocpip.org.

[17] See ST web site: http://www.stmcu.com/inchtml-pages-STBus_intro.html

[18] A. Greiner et al, "SPIN: a Scalable, Packet-switched, On-chip Micro-network, Proc. of Design Automation and Test in Europe (Designer Forum), Munich, March 2003.

[19] S.-J. Lee et al, "An 800MHz Star-Connected On-Chip Network for Application to Systems on a Chip", Proc. of Intl. Solid-State Circuits Conference (ISSC), San Francisco, Feb. 2003.

[20] M. Borgatti et al, "A 0.18um, 1GOPS Reconfigurable Signal Processing IC with embedded FPGA and 1.2GB/s, 3-Port Flash Memory Subsystem", Proc. of Intl. Solid-State Circuits Conference (ISSC), San Francisco, Feb. 2003.

[21] See EASIC Corp. web site: http://www.easic.com.

[22] P. G. Paulin, C. Pilkington, E. Bensoudane, M. Langevin, "A Multi-Processor SoC Platform and Tools for Communications Applications", in Embedded Systems Handbook, CRC Press, to appear in May 2004.

[23] E. Kohler et al., "The Click Modular Router," ACM Trans. Computer Systems, vol. 18, no. 3, Aug. 2000, pp. 263-297.

[24] N. Soni et al, "NPSE: A High Performance Network Packet Search Engine", Proc. of Design Automation and Test in Europe (Designer Forum), Munich, March 2003.